

# On the Shoshan-Zwick Algorithm for the All-Pairs Shortest Path Problem

Pavlos Eirinakis \*

Athens University of Economics and Business,  
Athens, Greece  
{peir@aueb.gr}

Matthew Williamson †

West Virginia University Institute of Technology  
Montgomery, WV, USA  
{matthew.williamson@mail.wvu.edu}

K. Subramani †

West Virginia University,  
Morgantown, WV, USA  
{ksmani@csee.wvu.edu}

## Abstract

The Shoshan-Zwick algorithm solves the all pairs shortest paths problem in undirected graphs with integer edge costs in the range  $\{1, 2, \dots, M\}$ . It runs in  $\tilde{O}(M \cdot n^\omega)$  time, where  $n$  is the number of vertices,  $M$  is the largest integer edge cost, and  $\omega < 2.3727$  is the exponent of matrix multiplication. It is the fastest known algorithm for this problem. This paper points out the erroneous behavior of the Shoshan-Zwick algorithm and revises the algorithm to resolve the issues that cause this behavior. Moreover, it discusses implementation aspects of the Shoshan-Zwick algorithm using currently-existing sub-cubic matrix multiplication algorithms.

## 1 Introduction

The Shoshan-Zwick algorithm [22] solves the all-pairs shortest paths (APSP) problem in undirected graphs, where the edge costs are integers in the range  $\{1, 2, \dots, M\}$ . This is accomplished by computing  $O(\log(M \cdot n))$  distance products of  $n \times n$  matrices with elements in the range  $\{1, 2, \dots, M\}$ . The algorithm runs in  $\tilde{O}(M \cdot n^\omega)$  time, where  $\omega < 2.3727$  is the exponent for the fastest known matrix multiplication algorithm [29].

The APSP problem is a fundamental problem in algorithmic graph theory. Consider a graph with  $n$  nodes and  $m$  edges. For directed or undirected graphs with real edge costs, we can use known methods that run in  $O(m \cdot n + n^2 \cdot \log n)$  time [8, 11, 17] and  $O(n^3)$  time [10]. Sub-cubic APSP algorithms have been obtained that run in  $O(n^3 \cdot ((\log \log n)/\log n)^{1/2})$  time [12],  $O(n^3 \cdot \sqrt{\log \log n / \log n})$  time [24],  $O(n^3 / \sqrt{\log n})$  time [9],  $O(n^3 \cdot (\log \log n / \log n)^{5/7})$  time [15],  $O(n^3 \cdot (\log \log n)^2 \log n)$  time [25],  $O(n^3 \cdot \log \log n / \log n)$  time [26],  $O(n^3 \cdot \sqrt{\log \log n} / \log n)$  time [31],  $O(n^3 / \log n)$  time [3], and  $O(n^3 \cdot \log \log n / \log^2 n)$  time [16]. For

\*This research has been co-financed by the European Union (European Social Fund - ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) - Research Funding Program: Thales. Investing in knowledge society through the European Social Fund. (MIS: 380232)

†This research was supported in part by the National Science Foundation and Air Force of Scientific Research through Awards CNS-0849735, CCF-1305054, and FA9550-12-1-0199.

undirected graphs with integer edge costs, the problem can be solved in  $O(m \cdot n)$  time [27, 28]. Fast matrix multiplication algorithms can also be used for solving the APSP problem in dense graphs with small integer edge costs. [13, 21] provide algorithms that run in  $\tilde{O}(n^\omega)$  time in unweighted, undirected graphs, where  $\omega < 2.3727$  is the exponent for matrix multiplication [29]. [4] improves this result with an algorithm that runs in  $o(m \cdot n)$  time.

In this paper, we revise the Shoshan-Zwick algorithm to resolve some issues that cause the algorithm to behave erroneously. This behavior was identified when, in the process of implementing the algorithm as part of a more elaborate procedure (i.e., identifying negative cost cycles in undirected graphs), we discovered that the algorithm is not producing correct results. Since the Shoshan-Zwick algorithm is incorrect in its current version, any result that uses this algorithm as a subroutine is also incorrect. For instance, the results provided by Alon and Yuster [2], Cygan et. al [7], W. Liu et al [19], and Yuster [30] all depend on the Shoshan-Zwick algorithm. Thus, their results are currently incorrect. By applying our revision to the algorithm, the issues with the above results are resolved. We also discuss issues concerning the implementation of the Shoshan-Zwick algorithm using known sub-cubic matrix multiplication algorithms.

The principal contributions of this paper are as follows:

- (i) A counter-example that shows that the Shoshan-Zwick algorithm is incorrect in its current form.
- (ii) A detailed explanation of where and why the algorithm fails.
- (iii) A modified version of the Shoshan-Zwick algorithm that corrects the problems with the previous version. The corrections do not affect the time complexity of the algorithm.
- (iv) A discussion concerning implementing the matrix multiplication subroutine that is used in the algorithm.

The rest of the paper is organized as follows. We describe the Shoshan-Zwick algorithm in Section 2. Section 3 establishes that the published version of the algorithm is incorrect by providing a simple counter-example. The origins of this erroneous behavior is identified in Section 4. In Section 5, we present a revised version of the algorithm and provide a formal proof of correctness. In Section 6, we discuss the efficacy of the Shoshan-Zwick algorithm. Section 7 concludes the paper by summarizing our contributions and discussing avenues for future research.

## 2 The Shoshan-Zwick Algorithm

In this section, we review the Shoshan-Zwick algorithm for the APSP problem in undirected graphs with integer edge costs. Consider a graph  $G = (V, E)$ , where  $V = \{0, 1, 2, \dots, n\}$  is the set of nodes, and  $E$  is the set of edges. The graph is represented as an  $n \times n$  matrix  $\mathbf{D}$ , where  $d_{ij}$  is the cost of edge  $(i, j)$  if  $(i, j) \in E$ ,  $d_{ii} = 0$  for  $1 \leq i \leq n$ , and  $d_{ij} = +\infty$  otherwise. Note that, without loss of generality, the edge costs are taken from the range  $\{1, 2, \dots, M\}$ , where  $M = 2^m$  for some  $m \geq 1$ .

The algorithm computes a logarithmic number of distance products in order to determine the shortest paths. Let  $\mathbf{A}$  and  $\mathbf{B}$  be two  $n \times n$  matrices. Their distance product  $\mathbf{A} \star \mathbf{B}$  is an  $n \times n$  matrix such that:

$$(\mathbf{A} \star \mathbf{B})_{ij} = \min_{k=1}^n \{a_{ik} + b_{kj}\}, 1 \leq i, j \leq n.$$

The distance product of two  $n \times n$  matrices with elements within the range  $\{1, 2, \dots, M, +\infty\}$  can be computed in  $\tilde{O}(M \cdot n^\omega)$  time [1]. The distance product of two matrices whose finite elements are taken from  $\{1, 2, \dots, M\}$  is a matrix whose finite elements are taken from  $\{1, 2, \dots, 2 \cdot M\}$ . However, the Shoshan-Zwick algorithm is based on not allowing the range of elements in the matrices it uses to increase. Hence, if  $\mathbf{A}$  is an  $n \times n$  matrix, and  $a, b$  are two numbers such that  $a \leq b$ , the algorithm utilizes two operations, namely  $\text{clip}(\mathbf{A}, a, b)$  and  $\text{chop}(\mathbf{A}, a, b)$ , that produce corresponding  $n \times n$  matrices such that

$$\begin{aligned} (\text{clip}(\mathbf{A}, a, b))_{ij} &= \begin{cases} a & \text{if } a_{ij} < a \\ a_{ij} & \text{if } a \leq a_{ij} \leq b \\ +\infty & \text{if } a_{ij} > b \end{cases} \\ (\text{chop}(\mathbf{A}, a, b))_{ij} &= \begin{cases} a_{ij} & \text{if } a \leq a_{ij} \leq b \\ +\infty & \text{otherwise} \end{cases}. \end{aligned}$$

The algorithm also defines  $n \times n$  matrices  $\mathbf{A} \wedge \mathbf{B}$ ,  $\mathbf{A} \bar{\wedge} \mathbf{B}$ , and  $\mathbf{A} \vee \mathbf{B}$  such that

$$\begin{aligned} (\mathbf{A} \wedge \mathbf{B})_{ij} &= \begin{cases} a_{ij} & \text{if } b_{ij} < 0 \\ +\infty & \text{otherwise} \end{cases} \\ (\mathbf{A} \bar{\wedge} \mathbf{B})_{ij} &= \begin{cases} a_{ij} & \text{if } b_{ij} \geq 0 \\ +\infty & \text{otherwise} \end{cases} \\ (\mathbf{A} \vee \mathbf{B})_{ij} &= \begin{cases} a_{ij} & \text{if } a_{ij} \neq +\infty \\ b_{ij} & \text{if } a_{ij} = +\infty, b_{ij} \neq +\infty \\ +\infty & \text{if } a_{ij} = b_{ij} = +\infty \end{cases}. \end{aligned}$$

Finally, if  $\mathbf{C} = (c_{ij})$  and  $\mathbf{P} = (p_{ij})$  are matrices, the algorithm defines the Boolean matrices  $(\mathbf{C} \geq 0)$  and  $(0 \leq \mathbf{P} \leq M)$  such that

$$\begin{aligned} (\mathbf{C} \geq 0)_{ij} &= \begin{cases} 1 & \text{if } c_{ij} \geq 0 \\ 0 & \text{otherwise} \end{cases} \\ (0 \leq \mathbf{P} \leq M)_{ij} &= \begin{cases} 1 & \text{if } 0 \leq p_{ij} \leq M \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

Using the definitions above, the Shoshan-Zwick algorithm computes the shortest paths by finding the  $\lceil \log_2 n \rceil$  most significant bits of each distance and the remainder that each distance leaves modulo  $M$ . This provides enough information to reconstruct the distances. The procedure is shown in Algorithm 2.1, and the proof of correctness is given in [22, Lemma 3.6].

The algorithm computes  $O(\log n + \log M) = O(\log(M \cdot n))$  distance products of matrices, whose finite

elements are in the range  $\{0, 1, 2, \dots, 2 \cdot M\}$  or in the range  $\{-M, \dots, M\}$ . Note that each distance product can be reduced to a constant number of distance products of matrices with elements in the range  $\{1, 2, \dots, M\}$ . All other operations in the algorithm take  $O(n^2 \cdot (\log n + \log M))$  time. Therefore, the total running time of the algorithm is  $\tilde{O}(M \cdot n^\omega)$  time [22, Theorem 3.7].

**Function SHOSHAN-ZWICK-APSP( $\mathbf{D}$ )**

```

1:  $l = \lceil \log_2 n \rceil$ .
2:  $m = \log_2 M$ .
3: for ( $k = 1$  to  $m + 1$ ) do
4:    $\mathbf{D} = \text{clip}(\mathbf{D} \star \mathbf{D}, 0, 2 \cdot M)$ .
5: end for
6:  $\mathbf{A}_0 = \mathbf{D} - M$ .
7: for ( $k = 1$  to  $l$ ) do
8:    $\mathbf{A}_k = \text{clip}(\mathbf{A}_{k-1} \star \mathbf{A}_{k-1}, -M, M)$ .
9: end for
10:  $\mathbf{C}_l = -M$ .
11:  $\mathbf{P}_l = \text{clip}(\mathbf{D}, 0, M)$ .
12:  $\mathbf{Q}_l = +\infty$ .
13: for ( $k = l - 1$  down to  $0$ ) do
14:    $\mathbf{C}_k = [\text{clip}(\mathbf{P}_{k+1} \star \mathbf{A}_k, -M, M) \wedge \mathbf{C}_{k+1}] \vee [\text{clip}(\mathbf{Q}_{k+1} \star \mathbf{A}_k, -M, M) \bar{\wedge} \mathbf{C}_{k+1}]$ .
15:    $\mathbf{P}_k = \mathbf{P}_{k+1} \vee \mathbf{Q}_{k+1}$ .
16:    $\mathbf{Q}_k = \text{chop}(\mathbf{C}_k, 1 - M, M)$ .
17: end for
18: for ( $k = 1$  to  $l$ ) do
19:    $\mathbf{B}_k = (\mathbf{C}_k \geq 0)$ .
20: end for
21:  $\mathbf{B}_0 = (0 \leq \mathbf{P}_0 < M)$ .
22:  $\mathbf{R} = \mathbf{P}_0 \bmod M$ .
23:  $\Delta = M \cdot \sum_{k=0}^l 2^k \cdot \mathbf{B}_k + \mathbf{R}$ .
24: return  $\Delta$ .
```

**Algorithm 2.1:** Shoshan-Zwick APSP Algorithm

### 3 A Counter-Example

In this section, we provide a detailed presentation of the counter-example presented that shows that the Shoshan-Zwick algorithm is incorrect as presented in [22]. Consider the APSP problem with respect to graph  $G'$  presented in Figure 1.

Graph  $G'$  can also be represented as a  $3 \times 3$  matrix:

$$\mathbf{D} = \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & \infty \\ 4 & \infty & 0 \end{bmatrix}.$$

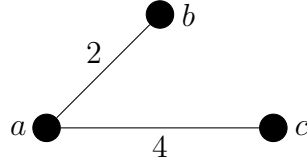


Figure 1: Graph  $G'$  of the counter-example for the Shoshan-Zwick algorithm.

We now walk through each step of the algorithm. First,  $l = \lceil \log_2 3 \rceil = 2$ , and  $m = \log_2 4 = 2$ . This means that in the **for** loop in lines 3 to 5, we set  $\mathbf{D} = \text{clip}(\mathbf{D} \star \mathbf{D}, 0, 8)$  three times. At each such step, we get

$$\begin{aligned}
 \mathbf{D} \star \mathbf{D} &= \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & \infty \\ 4 & \infty & 0 \end{bmatrix} \star \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & \infty \\ 4 & \infty & 0 \end{bmatrix} = \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & 6 \\ 4 & 6 & 0 \end{bmatrix}, & \mathbf{D} &= \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & 6 \\ 4 & 6 & 0 \end{bmatrix} \text{ (for } k=1), \\
 \mathbf{D} \star \mathbf{D} &= \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & 6 \\ 4 & 6 & 0 \end{bmatrix} \star \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & 6 \\ 4 & 6 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & 6 \\ 4 & 6 & 0 \end{bmatrix}, & \mathbf{D} &= \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & 6 \\ 4 & 6 & 0 \end{bmatrix} \text{ (for } k=2), \\
 \mathbf{D} \star \mathbf{D} &= \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & 6 \\ 4 & 6 & 0 \end{bmatrix} \star \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & 6 \\ 4 & 6 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & 6 \\ 4 & 6 & 0 \end{bmatrix}, & \mathbf{D} &= \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & 6 \\ 4 & 6 & 0 \end{bmatrix} \text{ (for } k=3).
 \end{aligned}$$

The next step is to set  $\mathbf{A}_0 = \mathbf{D} - \mathbf{M}$ , which gives us

$$\mathbf{A}_0 = \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & 6 \\ 4 & 6 & 0 \end{bmatrix} - 4 = \begin{bmatrix} -4 & -2 & 0 \\ -2 & -4 & 2 \\ 0 & 2 & -4 \end{bmatrix}.$$

In the **for** loop in lines 7 to 9, we compute  $\mathbf{A}_k = \text{clip}(\mathbf{A}_{k-1} \star \mathbf{A}_{k-1}, -4, 4)$  for  $k = 1$  and  $k = 2$ . This gives us

$$\begin{aligned}
 \mathbf{A}_0 \star \mathbf{A}_0 &= \begin{bmatrix} -4 & -2 & 0 \\ -2 & -4 & 2 \\ 0 & 2 & -4 \end{bmatrix} \star \begin{bmatrix} -4 & -2 & 0 \\ -2 & -4 & 2 \\ 0 & 2 & -4 \end{bmatrix} = \begin{bmatrix} -8 & -6 & -4 \\ -6 & -8 & -2 \\ -4 & -2 & -8 \end{bmatrix}, \mathbf{A}_1 = \begin{bmatrix} -4 & -4 & -4 \\ -4 & -4 & -2 \\ -4 & -2 & -4 \end{bmatrix}, \\
 \mathbf{A}_1 \star \mathbf{A}_1 &= \begin{bmatrix} -4 & -4 & -4 \\ -4 & -4 & -2 \\ -4 & -2 & -4 \end{bmatrix} \star \begin{bmatrix} -4 & -4 & -4 \\ -4 & -4 & -2 \\ -4 & -2 & -4 \end{bmatrix} = \begin{bmatrix} -8 & -8 & -8 \\ -8 & -8 & -8 \\ -8 & -8 & -8 \end{bmatrix}, \mathbf{A}_2 = \begin{bmatrix} -4 & -4 & -4 \\ -4 & -4 & -4 \\ -4 & -4 & -4 \end{bmatrix}.
 \end{aligned}$$

In lines 10 to 12, we set  $\mathbf{C}_2 = -4$ ,  $\mathbf{P}_2 = \text{clip}(\mathbf{D}, 0, 4)$ , and  $\mathbf{Q}_2 = +\infty$ . Thus, we have

$$\mathbf{C}_2 = \begin{bmatrix} -4 & -4 & -4 \\ -4 & -4 & -4 \\ -4 & -4 & -4 \end{bmatrix}, \mathbf{P}_2 = \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & \infty \\ 4 & \infty & 0 \end{bmatrix}, \mathbf{Q}_2 = \begin{bmatrix} \infty & \infty & \infty \\ \infty & \infty & \infty \\ \infty & \infty & \infty \end{bmatrix}.$$

We now compute the **for** loop in lines 13 to 17. This means we run the lines

$$\mathbf{C}_k = [\text{clip}(\mathbf{P}_{k+1} \star \mathbf{A}_k, -4, 4) \bigwedge \mathbf{C}_{k+1}] \bigvee [\text{clip}(\mathbf{Q}_{k+1} \star \mathbf{A}_k, -4, 4) \bar{\bigwedge} \mathbf{C}_{k+1}],$$

$$\mathbf{P}_k = \mathbf{P}_{k+1} \bigvee \mathbf{Q}_{k+1}, \text{ and}$$

$$\mathbf{Q}_k = \text{chop}(\mathbf{C}_k, -3, 4)$$

twice, i.e., for  $k = 1$  and  $k = 0$ . After this loop, we get

$$(\text{for } k = 1) \quad \mathbf{P}_2 \star \mathbf{A}_1 = \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & \infty \\ 4 & \infty & 0 \end{bmatrix} \star \begin{bmatrix} -4 & -4 & -4 \\ -4 & -4 & -2 \\ -4 & -2 & -4 \end{bmatrix} = \begin{bmatrix} -4 & -4 & -4 \\ -4 & -4 & -2 \\ -4 & -2 & -4 \end{bmatrix},$$

$$\text{clip}(\mathbf{P}_2 \star \mathbf{A}_1, -4, 4) = \begin{bmatrix} -4 & -4 & -4 \\ -4 & -4 & -2 \\ -4 & -2 & -4 \end{bmatrix},$$

$$\text{clip}(\mathbf{P}_2 \star \mathbf{A}_1, -4, 4) \bigwedge \mathbf{C}_2 = \begin{bmatrix} -4 & -4 & -4 \\ -4 & -4 & -2 \\ -4 & -2 & -4 \end{bmatrix} \bigwedge \begin{bmatrix} -4 & -4 & -4 \\ -4 & -4 & -4 \\ -4 & -4 & -4 \end{bmatrix} = \begin{bmatrix} -4 & -4 & -4 \\ -4 & -4 & -2 \\ -4 & -2 & -4 \end{bmatrix},$$

$$\mathbf{Q}_2 \star \mathbf{A}_1 = \begin{bmatrix} \infty & \infty & \infty \\ \infty & \infty & \infty \\ \infty & \infty & \infty \end{bmatrix} \star \begin{bmatrix} -4 & -4 & -4 \\ -4 & -4 & -2 \\ -4 & -2 & -4 \end{bmatrix} = \begin{bmatrix} \infty & \infty & \infty \\ \infty & \infty & \infty \\ \infty & \infty & \infty \end{bmatrix},$$

$$\text{clip}(\mathbf{Q}_2 \star \mathbf{A}_1, -4, 4) = \begin{bmatrix} \infty & \infty & \infty \\ \infty & \infty & \infty \\ \infty & \infty & \infty \end{bmatrix},$$

$$\text{clip}(\mathbf{Q}_2 \star \mathbf{A}_1, -4, 4) \bar{\bigwedge} \mathbf{C}_2 = \begin{bmatrix} \infty & \infty & \infty \\ \infty & \infty & \infty \\ \infty & \infty & \infty \end{bmatrix} \bar{\bigwedge} \begin{bmatrix} -4 & -4 & -4 \\ -4 & -4 & -4 \\ -4 & -4 & -4 \end{bmatrix} = \begin{bmatrix} \infty & \infty & \infty \\ \infty & \infty & \infty \\ \infty & \infty & \infty \end{bmatrix},$$

$$\mathbf{C}_1 = \begin{bmatrix} -4 & -4 & -4 \\ -4 & -4 & -2 \\ -4 & -2 & -4 \end{bmatrix} \bigvee \begin{bmatrix} \infty & \infty & \infty \\ \infty & \infty & \infty \\ \infty & \infty & \infty \end{bmatrix} = \begin{bmatrix} -4 & -4 & -4 \\ -4 & -4 & -2 \\ -4 & -2 & -4 \end{bmatrix},$$

$$\mathbf{P}_1 = \mathbf{P}_2 \bigvee \mathbf{Q}_2 = \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & \infty \\ 4 & \infty & 0 \end{bmatrix} \bigvee \begin{bmatrix} \infty & \infty & \infty \\ \infty & \infty & \infty \\ \infty & \infty & \infty \end{bmatrix} = \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & \infty \\ 4 & \infty & 0 \end{bmatrix},$$

$$\mathbf{Q}_1 = \text{chop}(\mathbf{C}_1, -3, 4) = \begin{bmatrix} \infty & \infty & \infty \\ \infty & \infty & -2 \\ \infty & -2 & \infty \end{bmatrix},$$

$$(\text{and for } k = 0) \quad \mathbf{P}_1 \star \mathbf{A}_0 = \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & \infty \\ 4 & \infty & 0 \end{bmatrix} \star \begin{bmatrix} -4 & -2 & 0 \\ -2 & -4 & 2 \\ 0 & 2 & -4 \end{bmatrix} = \begin{bmatrix} -4 & -2 & 0 \\ -2 & -4 & 2 \\ 0 & 2 & -4 \end{bmatrix},$$

$$\text{clip}(\mathbf{P}_1 \star \mathbf{A}_0, -4, 4) = \begin{bmatrix} -4 & -2 & 0 \\ -2 & -4 & 2 \\ 0 & 2 & -4 \end{bmatrix},$$

$$\begin{aligned}
clip(\mathbf{P}_1 \star \mathbf{A}_0, -4, 4) \bar{\wedge} \mathbf{C}_1 &= \begin{bmatrix} -4 & -2 & 0 \\ -2 & -4 & 2 \\ 0 & 2 & -4 \end{bmatrix} \bar{\wedge} \begin{bmatrix} -4 & -4 & -4 \\ -4 & -4 & -2 \\ -4 & -2 & -4 \end{bmatrix} = \begin{bmatrix} -4 & -2 & 0 \\ -2 & -4 & 2 \\ 0 & 2 & -4 \end{bmatrix}, \\
\mathbf{Q}_1 \star \mathbf{A}_0 &= \begin{bmatrix} \infty & \infty & \infty \\ \infty & \infty & -2 \\ \infty & -2 & \infty \end{bmatrix} \star \begin{bmatrix} -4 & -2 & 0 \\ -2 & -4 & 2 \\ 0 & 2 & -4 \end{bmatrix} = \begin{bmatrix} \infty & \infty & \infty \\ -2 & 0 & -6 \\ -4 & -6 & -4 \end{bmatrix}, \\
clip(\mathbf{Q}_1 \star \mathbf{A}_0, -4, 4) &= \begin{bmatrix} \infty & \infty & \infty \\ -2 & 0 & -4 \\ -4 & -4 & -4 \end{bmatrix}, \\
clip(\mathbf{Q}_1 \star \mathbf{A}_0, -4, 4) \bar{\wedge} \mathbf{C}_1 &= \begin{bmatrix} \infty & \infty & \infty \\ -2 & 0 & -4 \\ -4 & -4 & -4 \end{bmatrix} \bar{\wedge} \begin{bmatrix} -4 & -4 & -4 \\ -4 & -4 & -2 \\ -4 & -2 & -4 \end{bmatrix} = \begin{bmatrix} \infty & \infty & \infty \\ \infty & \infty & \infty \\ \infty & \infty & \infty \end{bmatrix}, \\
\mathbf{C}_0 &= \begin{bmatrix} -4 & -2 & 0 \\ -2 & -4 & 2 \\ 0 & 2 & -4 \end{bmatrix} \vee \begin{bmatrix} \infty & \infty & \infty \\ \infty & \infty & \infty \\ \infty & \infty & \infty \end{bmatrix} = \begin{bmatrix} -4 & -2 & 0 \\ -2 & -4 & 2 \\ 0 & 2 & -4 \end{bmatrix}, \\
\mathbf{P}_0 = \mathbf{P}_1 \vee \mathbf{Q}_1 &= \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & \infty \\ 4 & \infty & 0 \end{bmatrix} \vee \begin{bmatrix} \infty & \infty & \infty \\ \infty & \infty & -2 \\ \infty & -2 & \infty \end{bmatrix} = \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & -2 \\ 4 & -2 & 0 \end{bmatrix}, \\
\mathbf{Q}_0 = chop(\mathbf{C}_0, -3, 4) &= \begin{bmatrix} \infty & -2 & \infty \\ -2 & \infty & 2 \\ \infty & 2 & \infty \end{bmatrix}.
\end{aligned}$$

In the **for** loop in lines 18 to 20, we set  $\mathbf{B}_k = (\mathbf{C}_k \geq 0)$  for  $k = 1$  and  $k = 2$ . This means

$$\begin{aligned}
\mathbf{B}_1 = (\mathbf{C}_1 \geq 0) &= \left( \begin{bmatrix} -4 & -4 & -4 \\ -4 & -4 & -2 \\ -4 & -2 & -4 \end{bmatrix} \geq 0 \right) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \\
\mathbf{B}_2 = (\mathbf{C}_2 \geq 0) &= \left( \begin{bmatrix} -4 & -4 & -4 \\ -4 & -4 & -4 \\ -4 & -4 & -4 \end{bmatrix} \geq 0 \right) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}.
\end{aligned}$$

We then set  $\mathbf{B}_0 = (0 \leq \mathbf{P}_0 < 4)$ ,  $\mathbf{R} = \mathbf{P}_0 \bmod 4$ , and  $\Delta = 4 \cdot \sum_{k=0}^2 2^k \cdot \mathbf{B}_k + \mathbf{R}$  according to lines 21 to 23:

$$\begin{aligned}
\mathbf{B}_0 = (0 \leq \mathbf{P}_0 < 4) &= \left( 0 \leq \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & -2 \\ 4 & -2 & 0 \end{bmatrix} < 4 \right) = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \\
\mathbf{R} = \mathbf{P}_0 \bmod 4 &= \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & -2 \\ 4 & -2 & 0 \end{bmatrix} \bmod 4 = \begin{bmatrix} 0 & 2 & 0 \\ 2 & 0 & -2 \\ 0 & -2 & 0 \end{bmatrix},
\end{aligned}$$

$$\begin{aligned}
\Delta &= 4 \cdot \sum_{k=0}^2 2^k \cdot \mathbf{B}_k + \mathbf{R} \\
&= 4 \cdot \left( 2^0 \cdot \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + 2^1 \cdot \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} + 2^2 \cdot \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \right) + \begin{bmatrix} 0 & 2 & 0 \\ 2 & 0 & -2 \\ 0 & -2 & 0 \end{bmatrix} \\
&= 4 \cdot \left( \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \right) + \begin{bmatrix} 0 & 2 & 0 \\ 2 & 0 & -2 \\ 0 & -2 & 0 \end{bmatrix} \\
&= \begin{bmatrix} 4 & 4 & 0 \\ 4 & 4 & 0 \\ 0 & 0 & 4 \end{bmatrix} + \begin{bmatrix} 0 & 2 & 0 \\ 2 & 0 & -2 \\ 0 & -2 & 0 \end{bmatrix} = \begin{bmatrix} 4 & 6 & 0 \\ 6 & 4 & -2 \\ 0 & -2 & 4 \end{bmatrix}
\end{aligned}$$

The algorithm terminates by returning  $\Delta$  in line 24. The resulting shortest path costs are presented in Table 1. However, if we examine graph  $G'$  (Figure 1), we find that these results are *incorrect*. The correct shortest paths are provided in Table 2. Therefore, the Shoshan-Zwick algorithm is incorrect.

Table 1: Shortest paths for graph  $G'$  based on the Shoshan-Zwick algorithm.

$\Delta$	$a$	$b$	$c$
$a$	4	6	0
$b$	6	4	-2
$c$	0	-2	4

Table 2: Correct shortest paths for graph  $G'$ .

$\Delta$	$a$	$b$	$c$
$a$	0	2	4
$b$	2	0	6
$c$	4	6	0

## 4 The Errors in the Algorithm

In this section, we describe what causes the erroneous behavior of the Shoshan-Zwick algorithm. Recall that  $\Delta$  is the matrix that contains the costs of the shortest paths between all pairs of vertices after the algorithm terminates. Moreover, let  $\delta(i, j)$  denote the cost of the shortest path between nodes  $i$  and  $j$ . After the termination of the algorithm, we must have  $\Delta_{ij} = \delta(i, j)$  for any  $i, j \in \{1, \dots, n\}$ . However, as we showed in the counter-example in Section 3, it may be the case that  $\Delta_{ij} \neq \delta(i, j)$  for some  $i, j$  at termination. The exact errors of the algorithm are as follows:

1.  $\mathbf{R}$  is not computed correctly.
2.  $\mathbf{B}_0$  is not computed correctly.
3.  $\Delta$  is not computed correctly, since  $M \cdot \mathbf{B}_0 + \mathbf{R}$  is part of the sum producing it.

In the rest of this section, we illustrate what causes these errors.



It is clear from line 23 of Algorithm 2.1 that the matrices  $\mathbf{B}_k$  (for  $0 \leq k \leq l$ ) represent the  $\lceil \log_2 n \rceil$  most significant bits of each distance. That is,

$$(\mathbf{B}_k)_{ij} = \begin{cases} 1 & \text{if } 2^k \cdot M \text{ must be added to } \Delta_{ij} \text{ so that } \Delta_{ij} = \delta(i, j) \\ 0 & \text{otherwise} \end{cases},$$

while  $\mathbf{R}$  represents the remainder of each distance modulo  $M$ . This is also illustrated in [22, Lemma 3.6], where for every  $0 \leq k \leq l$ ,  $(\mathbf{B}_k)_{ij} = 1$  if and only if  $\delta(i, j) \bmod 2^{k+m+1} \geq 2^{k+m}$ , while  $\mathbf{R}_{ij} = \delta(i, j) \bmod M$ . Hence, for every  $i, j$ , we must have

$$(M \cdot \mathbf{B}_0 + \mathbf{R})_{ij} = \delta(i, j) \bmod 2^{m+1}. \quad (1)$$

The first error of the algorithm arises immediately from the key observation that  $\mathbf{P}_0$  can have entries with negative values. This means that line 22 (that sets  $\mathbf{R}_{ij} = (\mathbf{P}_0)_{ij} \bmod M$ ) is not correctly calculating  $\mathbf{R}_{ij} = \delta(i, j) \bmod M$  since  $\delta(i, j) \geq 0$  by definition, while  $(\mathbf{P}_0)_{ij}$  can be negative.

A closer examination of how  $\mathbf{P}_0$  obtains its negative values reveals another error of the algorithm. The following definitions are given in [22, Section 3]. Consider a set  $Y \subseteq [0, M \cdot n]$ . Note that  $[0, M \cdot n]$  includes any value that  $\delta(i, j)$  can take, since  $n$  is the number of nodes, and  $M$  is the maximum edge cost. Let  $Y = \cup_{r=1}^p [a_r, b_r]$ , where  $a_r \leq b_r$ , for  $1 \leq r \leq p$  and  $b_r < a_{r+1}$ , for  $1 \leq r < p$ . Let  $\Delta_Y$  be an  $n \times n$  matrix, whose elements are in the range  $\{-M, \dots, M\} \cup \{+\infty\}$ , such that for every  $1 \leq i, j \leq n$ , we have

$$(\Delta_Y)_{ij} = \begin{cases} -M & \text{if } a_r \leq \delta(i, j) \leq b_r - M \text{ for some } 1 \leq r \leq p, \\ \delta(i, j) - b_r & \text{if } b_r - M < \delta(i, j) \leq b_r + M \text{ for some } 1 \leq r \leq p, \\ +\infty & \text{otherwise.} \end{cases} \quad (2)$$

By [22, Lemma 3.5],  $\mathbf{P}_0 = \Delta_{Y_0}$ , where  $Y_0 = \{x | (x \bmod 2^{m+1}) = 0\}$ . Recall that  $2^m = M$ . Note that by definition of  $Y_0$ , when calculating  $\mathbf{P}_0 = \Delta_{Y_0}$ , it can only be the case that  $a_r = b_r$ . Moreover,  $b_r = 2^{m+1} \cdot (r-1)$  for  $1 \leq r \leq p$ , where  $p$  is such that  $2^{m+1} \cdot (p-1) \leq M \cdot n < 2^{m+1} \cdot p$ . But then:

$$(\cup_{r=1}^p [b_r - M, b_r + M]) \supset [0, M \cdot n]$$

That is,  $(\cup_{r=1}^p [b_r - M, b_r + M])$  covers all possible values that  $\delta(i, j)$  may take for any  $i, j$ . Hence,

$$(\mathbf{P}_0)_{ij} = \begin{cases} \delta(i, j) & \text{for } r = 1 \text{ (i.e., if } \delta(i, j) \leq 2^m), \\ \delta(i, j) - b_r & \text{for } 2 \leq r \leq p, \text{ such that } b_r - 2^m < \delta(i, j) \leq b_r + 2^m. \end{cases} \quad (3)$$

Let us examine the values that  $(\mathbf{P}_0)_{ij}$  takes by equation (3):

- For  $0 \leq \delta(i, j) \leq 2^m$ , we have  $(\mathbf{P}_0)_{ij} = \delta(i, j) \bmod 2^{m+1}$ .
- For  $2^m < \delta(i, j) < 2^m + 2^m$ , we have  $(\mathbf{P}_0)_{ij} = (\delta(i, j) \bmod 2^{m+1}) - 2^{m+1}$ .
- For  $2^{m+1} \leq \delta(i, j) \leq 2^{m+1} + 2^m$ , we have  $(\mathbf{P}_0)_{ij} = \delta(i, j) \bmod 2^{m+1}$ .
- For  $2^{m+1} + 2^m < \delta(i, j) < 2^{m+2} + 2^m$ , we have  $(\mathbf{P}_0)_{ij} = (\delta(i, j) \bmod 2^{m+1}) - 2^{m+1}$ .

◦ And so forth...

More formally, equation (3) can be rewritten as follows:

$$(\mathbf{P}_0)_{ij} = \begin{cases} \delta(i, j) \bmod 2^{m+1} & \text{if } \delta(i, j) \bmod 2^{m+1} \leq 2^m, \\ (\delta(i, j) \bmod 2^{m+1}) - 2^{m+1} & \text{if } \delta(i, j) \bmod 2^{m+1} > 2^m. \end{cases} \quad (4)$$

Moreover, equation (4) implies that

$$\text{for } i, j \text{ such that } \delta(i, j) \bmod 2^{m+1} \leq 2^m, \text{ we have } 0 \leq (\mathbf{P}_0)_{ij} \leq M, \quad (5)$$

while

$$\text{for } i, j \text{ such that } \delta(i, j) \bmod 2^{m+1} > 2^m, \text{ we have } -M < (\mathbf{P}_0)_{ij} < 0. \quad (6)$$

Recall now that we must have  $(\mathbf{B}_0)_{ij} = 1$  if and only if  $\delta(i, j) \bmod 2^{m+1} \geq 2^m$ . However, from equations (5) and (6), this does not hold (as claimed in the proof of [22, Lemma 3.6]) for  $\mathbf{B}_0 = (0 \leq \mathbf{P}_0 < M)$  (i.e., line 21 of Algorithm 2.1). Therefore, the algorithm does not compute  $\mathbf{B}_0$  correctly.

It is clear that in the presence of these two identified errors (in calculating  $\mathbf{R}$  and  $\mathbf{B}_0$ ), the algorithm is not computing  $\Delta$  correctly.

To accommodate understanding, let us consider the case of  $\mathbf{P}_0$  for graph  $G'$  (Figure 1) in Section 3. We have  $Y_0 \subseteq [0, M \cdot n]$ , i.e.,  $Y_0 \subseteq [0, 12]$ . Further,  $Y_0 = \{x \mid (x \bmod 2^{2+1}) = 0\}$ . This means that  $Y_0 = \{0, 8\}$ . Thus, with respect to the definition of  $(\Delta_Y)_{ij}$ , we have  $a_1 = b_1 = 0$  and  $a_2 = b_2 = 8$ . Therefore,

$$(\mathbf{P}_0)_{ij} = (\Delta_{Y_0})_{ij} = \begin{cases} \delta(i, j) & \text{if } -4 < \delta(i, j) \leq 4 \text{ (for } r = 1 \text{ and hence } b_r = 0), \\ \delta(i, j) - 8 & \text{if } 4 < \delta(i, j) \leq 12 \text{ (for } r = 2 \text{ and hence } b_r = 8). \end{cases}$$

Let us consider the shortest path cost of nodes  $a$  ( $i = 1$ ) and  $b$  ( $j = 2$ ). We have  $\delta(1, 2) = 2$ , and thus  $(\mathbf{P}_0)_{12} = 2$ . We now consider the shortest path cost of nodes  $b$  ( $i = 2$ ) and  $c$  ( $j = 3$ ). We then have  $\delta(2, 3) = 6$ . This, however, means that  $(\mathbf{P}_0)_{23} = -2$ . Although  $\delta(2, 3) \bmod 2^3 > 2^2$ , we have  $(\mathbf{B}_0)_{23} = 0$ . Moreover,  $\mathbf{R}_{23} = (\mathbf{P}_0)_{23} \bmod 8 = -2$  (instead of  $\mathbf{R}_{23} = \delta(2, 3) \bmod 8 = 6$ ). These two errors lead to  $\Delta_{23} = -2$  instead of 6 (see Table 2).

## 5 The Revised Algorithm

In this section, we present a new version of the Shoshan-Zwick algorithm that resolves the problems illustrated in Section 4. Lines 1 to 20 of Algorithm 2.1 remain unchanged. We make the following changes to lines 21 to 24:

1. We replace  $\mathbf{B}_0$  with  $\hat{\mathbf{B}}_0$  and set  $\hat{\mathbf{B}}_0$  to  $(-M < \mathbf{P}_0 < 0)$  in line 21.
2. We replace  $\mathbf{R}$  with  $\hat{\mathbf{R}}$  and set  $\hat{\mathbf{R}}$  to  $\mathbf{P}_0$  in line 22.

3. We set  $\Delta$  to  $M \cdot \sum_{k=1}^l 2^k \cdot \mathbf{B}_k + 2 \cdot M \cdot \hat{\mathbf{B}}_0 + \hat{\mathbf{R}}$  in line 23.

Note that we have replaced  $\mathbf{B}_0$  and  $\mathbf{R}$  with  $\hat{\mathbf{B}}_0$  and  $\hat{\mathbf{R}}$ , respectively. The purpose of the change in notation is to show that these matrices no longer represent the incorrect versions from the original (erroneous) algorithm. Lines 21 to 24 of the revised algorithm are illustrated in Algorithm 5.1.

21:  $\hat{\mathbf{B}}_0 = (-M < \mathbf{P}_0 < 0)$ .  
 22:  $\hat{\mathbf{R}} = \mathbf{P}_0$ .  
 23:  $\Delta = M \cdot \sum_{k=1}^l 2^k \cdot \mathbf{B}_k + 2 \cdot M \cdot \hat{\mathbf{B}}_0 + \hat{\mathbf{R}}$ .  
 24: **return**  $\Delta$ .

**Algorithm 5.1:** Corrected portion of lines 21 to 24.

We refer to our counter-example in Section 3. Since the algorithm is correct up to line 20, we will examine how the algorithm operates in lines 21 to 24. Hence, we set  $\hat{\mathbf{B}}_0 = (-4 < \mathbf{P}_0 < 0)$ ,  $\hat{\mathbf{R}} = \mathbf{P}_0$ , and  $\Delta = 4 \cdot \sum_{k=1}^2 2^k \cdot \mathbf{B}_k + 8 \cdot \hat{\mathbf{B}}_0 + \hat{\mathbf{R}}$ .

$$\hat{\mathbf{B}}_0 = (-4 < \mathbf{P}_0 < 0) = \left( -4 \leq \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & -2 \\ 4 & -2 & 0 \end{bmatrix} < 0 \right) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix},$$

$$\hat{\mathbf{R}} = \mathbf{P}_0 = \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & -2 \\ 4 & -2 & 0 \end{bmatrix},$$

$$\begin{aligned} \Delta &= 4 \cdot \sum_{k=1}^2 2^k \cdot \mathbf{B}_k + 8 \cdot \hat{\mathbf{B}}_0 + \hat{\mathbf{R}} \\ &= 4 \cdot \left( 2^1 \cdot \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} + 2^2 \cdot \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \right) + 8 \cdot \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & -2 \\ 4 & -2 & 0 \end{bmatrix} \\ &= \begin{bmatrix} 0 & 2 & 4 \\ 2 & 0 & 6 \\ 4 & 6 & 0 \end{bmatrix} \end{aligned}$$

Line 24 returns  $\Delta$ . The elements of  $\Delta$  reflect the correct shortest path costs given in Table 2. Therefore, the revised algorithm works correctly for our counter-example. The corrections in the algorithm are not limited to the counter-example, as shown in the theorem below:

**Theorem 5.1** *The revised Shoshan-Zwick algorithm calculates all the shortest path costs in an undirected graph with integer edge costs in the range  $\{1, \dots, M\}$ .*

**Proof.** It suffices to show that  $2 \cdot M \cdot \hat{\mathbf{B}}_0 + \hat{\mathbf{R}}$  represents what the original algorithm intended to represent with  $M \cdot \mathbf{B}_0 + \mathbf{R}$ . That is, by equation (1), it suffices to show that  $(2 \cdot M \cdot \hat{\mathbf{B}}_0 + \hat{\mathbf{R}})_{ij} = \delta(i, j) \mod 2^{m+1}$ , for every  $1 \leq i, j \leq n$ .

First, we consider the case where  $\delta(i, j) \mod 2^{m+1} \leq 2^m$ . Equation (5) indicates that  $0 \leq (\mathbf{P}_0)_{ij} \leq M$ . Hence,  $(\hat{\mathbf{B}}_0)_{ij} = 0$  (by line 21 of the revised algorithm). Moreover, since  $\hat{\mathbf{R}}_{ij} = (\mathbf{P}_0)_{ij}$  (by line 22 of the

revised algorithm), we have that  $\hat{\mathbf{R}}_{ij} = \delta(i, j) \bmod 2^{m+1}$  by equation (4). Thus,  $(2 \cdot M \cdot \hat{\mathbf{B}}_0 + \hat{\mathbf{R}})_{ij} = \delta(i, j) \bmod 2^{m+1}$ .

We next consider the case where  $\delta(i, j) \bmod 2^{m+1} > 2^m$ . Equation (6) indicates that  $-M < (\mathbf{P}_0)_{ij} < 0$ . Hence,  $(\hat{\mathbf{B}}_0)_{ij} = 1$  (by line 21 of the revised algorithm). Further,  $\hat{\mathbf{R}}_{ij} = (\delta(i, j) \bmod 2^{m+1}) - 2^{m+1}$  by equation (4). Therefore,  $(2 \cdot M \cdot \hat{\mathbf{B}}_0 + \hat{\mathbf{R}})_{ij} = 2 \cdot 2^m \cdot 1 + (\delta(i, j) \bmod 2^{m+1}) - 2^{m+1} = \delta(i, j) \bmod 2^{m+1}$ , which completes the proof.  $\square$

## 6 Efficacy

In this section, we identify issues with implementing the Shoshan-Zwick algorithm. Recall that the algorithm runs in  $\tilde{O}(M \cdot n^\omega)$  time, where  $\omega$  is the exponent for the fastest known matrix multiplication algorithm. This means that the running time of the algorithm depends on which matrix multiplication algorithm is used. We will discuss two subcubic matrix multiplication algorithms and show why it is impractical to use them in the Shoshan-Zwick implementation.

The current fastest matrix multiplication algorithm is provided by [29], where  $\omega < 2.3727$ . This approach tightens the techniques used in [5], which gives a matrix multiplication algorithm where  $\omega < 2.376$ . Although both matrix multiplication algorithms are theoretically efficient, neither one is practical to implement. They both provide an advantage only for matrices that are too large for modern hardware to handle [20].

We next consider Strassen's matrix multiplication algorithm [23], which runs in  $O(n^{2.8074})$  time. Recall that the algorithm computes  $\mathbf{A} \cdot \mathbf{B} = \mathbf{C}$  by partitioning the matrices into equally sized block matrices

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} \end{bmatrix}, \mathbf{B} = \begin{bmatrix} \mathbf{B}_{1,1} & \mathbf{B}_{1,2} \\ \mathbf{B}_{2,1} & \mathbf{B}_{2,2} \end{bmatrix}, \mathbf{C} = \begin{bmatrix} \mathbf{C}_{1,1} & \mathbf{C}_{1,2} \\ \mathbf{C}_{2,1} & \mathbf{C}_{2,2} \end{bmatrix}$$

where

$$\begin{aligned} \mathbf{C}_{1,1} &= \mathbf{A}_{1,1} \cdot \mathbf{B}_{1,1} + \mathbf{A}_{1,2} \cdot \mathbf{B}_{2,1} \\ \mathbf{C}_{1,2} &= \mathbf{A}_{1,1} \cdot \mathbf{B}_{1,2} + \mathbf{A}_{1,2} \cdot \mathbf{B}_{2,2} \\ \mathbf{C}_{2,1} &= \mathbf{A}_{2,1} \cdot \mathbf{B}_{1,1} + \mathbf{A}_{2,2} \cdot \mathbf{B}_{2,1} \\ \mathbf{C}_{2,2} &= \mathbf{A}_{2,1} \cdot \mathbf{B}_{1,2} + \mathbf{A}_{2,2} \cdot \mathbf{B}_{2,2} \end{aligned}$$

To reduce the total number of multiplications, seven new matrices are defined as follows:

$$\begin{aligned} \mathbf{M}_1 &= (\mathbf{A}_{1,1} + \mathbf{A}_{2,2}) \cdot (\mathbf{B}_{1,1} + \mathbf{B}_{2,2}) \\ \mathbf{M}_2 &= (\mathbf{A}_{2,1} + \mathbf{A}_{2,2}) \cdot \mathbf{B}_{1,1} \\ \mathbf{M}_3 &= \mathbf{A}_{1,1} \cdot (\mathbf{B}_{1,2} - \mathbf{B}_{2,2}) \\ \mathbf{M}_4 &= \mathbf{A}_{2,2} \cdot (\mathbf{B}_{2,1} - \mathbf{B}_{1,1}) \\ \mathbf{M}_5 &= (\mathbf{A}_{1,1} + \mathbf{A}_{1,2}) \cdot \mathbf{B}_{2,2} \\ \mathbf{M}_6 &= (\mathbf{A}_{2,1} - \mathbf{A}_{1,1}) \cdot (\mathbf{B}_{1,1} + \mathbf{B}_{1,2}) \\ \mathbf{M}_7 &= (\mathbf{A}_{1,2} - \mathbf{A}_{2,2}) \cdot (\mathbf{B}_{2,1} + \mathbf{B}_{2,2}) \end{aligned}$$

With these new matrices, the block matrices of  $\mathbf{C}$  can be redefined as

$$\begin{aligned}\mathbf{C}_{1,1} &= \mathbf{M}_1 + \mathbf{M}_4 - \mathbf{M}_5 + \mathbf{M}_7 \\ \mathbf{C}_{1,2} &= \mathbf{M}_3 + \mathbf{M}_5 \\ \mathbf{C}_{2,1} &= \mathbf{M}_2 + \mathbf{M}_4 \\ \mathbf{C}_{2,2} &= \mathbf{M}_1 - \mathbf{M}_2 + \mathbf{M}_3 + \mathbf{M}_6\end{aligned}$$

The process of dividing  $\mathbf{C}$  repeats recursively  $n$  times until the submatrices degenerate into a single number.

Although Strassen's algorithm is faster than the naive  $O(n^3)$  matrix multiplication algorithm, we cannot directly use it in the Shoshan-Zwick algorithm. Recall that the naive approach uses matrix multiplication over the closed semi-ring  $\{+, \cdot\}$ . The Shoshan-Zwick algorithm, on the other hand, actually uses matrix multiplication over the closed semi-ring  $\{\min, +\}$ , which is known as “funny matrix multiplication” or the “distance product” in the literature. Note that the sum operation in the naive approach is equivalent to the min operation in “funny matrix multiplication”. However, Strassen's algorithm requires an additive inverse. This implies that an inverse for the min operation is needed in “funny matrix multiplication”. Such an inverse does not exist. In fact, we cannot multiply matrices with less than  $\Omega(n^3)$  operations when only the min and sum operations are allowed [1, 18]. Thus, Strassen's algorithm cannot directly be used for computing shortest paths.

One potential solution is to encode a matrix used for distance products such that regular matrix multiplication works. [1] provides an approach for this conversion as follows: Suppose we want to convert an  $n \times n$  matrix  $\mathbf{A}$  to  $\mathbf{A}'$ . We set

$$a'_{ij} = (n + 1)^{a_{ij} - x},$$

where  $x$  is the smallest value in  $\mathbf{A}$ . We perform the same conversion from  $\mathbf{B}$  to  $\mathbf{B}'$ . We then obtain  $\mathbf{C}' = \mathbf{A}' \cdot \mathbf{B}'$  by:

$$c'_{ij} = \sum_{k=1}^n (n + 1)^{a_{ik} + b_{kj} - 2 \cdot x}.$$

We then use binary search to find the largest  $s_{ij}$  such that  $s_{ij} \leq a'_{ik} + b'_{kj} - 2 \cdot x$ , and we set  $c_{ij} = s_{ij} + 2 \cdot x$ . This gives us  $\mathbf{C}$ , which is the distance product of matrices  $\mathbf{A}$  and  $\mathbf{B}$ .

[1] states that the algorithm performs  $O(n^\omega)$  operations on integers which are  $\leq n \cdot (n + 1)^{2 \cdot M}$ , where  $M$  is the magnitude of the largest number. For large numbers, we would need  $O(M \cdot \log M)$  operations on  $O(\log n)$ -bit numbers. As a result, the total time to compute the distance product is  $O(M \cdot n^\omega \cdot \log M)$ . If we apply this to the Shoshan-Zwick algorithm, the algorithm takes  $\tilde{O}(M^2 \cdot n^\omega \cdot \log M)$  time.

Although the above algorithm provides a subcubic approach for implementing the Shoshan-Zwick algorithm, it is not necessarily the most efficient implementation. This is because there exist other efficient APSP algorithms for integer edge costs. For instance, we can implement Goldberg's  $O(m + n \cdot \log N)$  time single source shortest path algorithm [14], where  $N$  is the largest edge cost, and run it  $n$  times; one for each vertex. Goldberg's implementation is one of the currently known fastest implementations available. The resulting implementation is substantially faster compared to the Shoshan-Zwick algorithm.

## 7 Conclusion

In this paper, we revised the Shoshan-Zwick algorithm to resolve issues related to its correctness. We first provided a counter-example which shows that the algorithm is incorrect. We then identified the exact cause of the problem and presented a modified algorithm that resolves the problem. We also explained the efficacy issues of the algorithm. An interesting study would be to implement the Shoshan-Zwick algorithm and profile it with efficient APSP algorithms for graphs with integer edge costs.

## References

- [1] Noga Alon, Zvi Galil, and Oded Margalit. On the exponent of the all pairs shortest path problem. *J. Comput. Syst. Sci.*, 54(2):255–262, 1997.
- [2] Noga Alon and Raphael Yuster. Fast algorithms for maximum subset matching and all-pairs shortest paths in graphs with a (not so) small vertex cover. In *ESA*, pages 175–186, 2007.
- [3] Timothy M. Chan. All-pairs shortest paths with real weights in  $O(n^3 \log n)$  time. *Algorithmica*, 50(2):236–243, 2008.
- [4] Timothy M. Chan. All-pairs shortest paths for unweighted undirected graphs in  $o(mn)$  time. *ACM Transactions on Algorithms*, 8(4):34, 2012.
- [5] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, STOC '87, pages 1–6, New York, NY, USA, 1987. ACM.
- [6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009.
- [7] Marek Cygan, Harold N. Gabow, and Piotr Sankowski. Algorithmic applications of Baur-Strassen’s theorem: shortest cycles, diameter and matchings. In *FOCS*, pages 531–540, 2012.
- [8] E. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.
- [9] W. Dobosiewicz. A more efficient algorithm for the min-plus multiplication. *Int. J. Comput. Math.*, 32:251–280, 1990.
- [10] Robert W. Floyd. Algorithm 97: shortest path. *Communications of the ACM*, 5(6):345, 1962.
- [11] M. Fredman and R.E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34:596–615, 1987.
- [12] Michael L. Fredman. New bounds on the complexity of the shortest path problem. *SIAM J. Comput.*, 5(1):83–89, 1976.

- [13] Zvi Galil and Oded Margalit. All pairs shortest paths for graphs with small integer length edges. *Journal of Computer and System Sciences*, 54(2):243–254, 1997.
- [14] A.V. Goldberg. A simple shortest path algorithm with linear average time. In *9th Ann. European Symp. on Algorithms (ESA 2001)*, volume 2161 of *Lecture Notes in Comput. Sci.*, pages 230–241, Aachen, Germany, 2001. Springer.
- [15] Y. Han. Improved algorithm for all pairs shortest paths. *Inform. Process. Lett.*, 91(5):245–250, 2004.
- [16] Yijie Han and Tadao Takaoka. An  $O(n^3 \log \log n / \log^2 n)$  time algorithm for all pairs shortest paths. In *SWAT*, pages 131–141, 2012.
- [17] D. Johnson. Efficient algorithms for shortest paths in sparse graphs. *Journal of the ACM*, 24:1–13, 1977.
- [18] L.R. Kerr. The effect of algebraic structure on the computational complexity of matrix multiplication. Ph.D. Thesis, Cornell University, 1970.
- [19] W. Liu, D. Wang, H. Jiang, W. Liu, and Chonggang Wang. Approximate convex decomposition based localization in wireless sensor networks. In *Proc. of IEEE INFOCOM*, pages 1853–1861, 2012.
- [20] S. Robinson. Toward an optimal algorithm for matrix multiplication. *SIAM News*, 38(9), 2005.
- [21] Raimund Seidel. On the all-pairs-shortest-path problem in unweighted undirected graphs. *Journal of Computer and System Sciences*, 51(3):400–403, December 1995.
- [22] A. Shoshan and U. Zwick. All pairs shortest paths in undirected graphs with integer weights. In *FOCS*, pages 605–614, 1999.
- [23] V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 14(3):354–356, 1969.
- [24] Tadao Takaoka. A new upper bound on the complexity of the all pairs shortest path problem. *Inform. Process. Lett.*, 43(4):195–199, 1992.
- [25] Tadao Takaoka. A faster algorithm for the all-pairs shortest path problem and its application. In *COCOON*, pages 278–289, 2004.
- [26] Tadao Takaoka. An  $O(n^3 \log \log n / \log n)$  time algorithm for the all-pairs shortest path problem. *Inf. Process. Lett.*, 96(5):155–161, 2005.
- [27] Mikkel Thorup. Undirected single source shortest path in linear time. In *FOCS*, pages 12–21, 1997.
- [28] Mikkel Thorup. Floats, integers, and single source shortest paths. In *STACS*, pages 14–24, 1998.
- [29] Virginia Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. In *STOC*, pages 887–898, 2012.

- [30] Raphael Yuster. A shortest cycle for each vertex of a graph. *Inf. Process. Lett.*, 111(21-22):1057–1061, 2011.
- [31] Uri Zwick. A slightly improved sub-cubic algorithm for the all pairs shortest paths problem with real edge lengths. *Algorithmica*, 46(2):181–192, 2006.